2003-06

Experiments with Deceptive Software Responses to Buffer-Overflow Attacks

Julian, Donald P.

Monterey, California. Naval Postgraduate School

Experiments with Deceptive Software Responses to Buffer-Overflow Attacks

MAJ Donald P. Julian, Neil C. Rowe, and J. Bret Michael

Computer Science Department, U.S. Naval Postgraduate School
Code CS/Rp, 833 Dyer Road, Monterey, CA 93943
juliandp at mctssa.usmc.mil., ncrowe at nps.navy.mil, and bmichael at nps.navy.mil

Index terms ? deception, information systems, decoys, World Wide Web, portals, servlets, buffer overflows

EXTENDED ABSTRACT

Modern intrusion detection systems have become good in identifying many kinds of malicious users on computer systems. But once they identify an attack, their usual response is to terminate the attacker session. This tells the attacker that they have been discovered, and encourages them to try other perhaps more vulnerable sites or try attack methods that we have no protection against. But access control is not the only response possible to an attack. Systems could use deception to fool the attacker about the results of their actions so that the attacker would waste time on fruitless endeavors. Deceptive software could also provide autonomous protective software responses to identified intrusions for a "second line of defense" when access controls have been subverted or destroyed [1].

One approach to managing deception in software is to "instrument" an operating system by designing "wrappers" to control interfaces to critical parts of it [2]. The wrappers could communicate through a shared database of information about the attack to provide a consistent deception. But this requires modifying an entire operating system; this is much work, even with tools, and is generally only feasible when source code is available, which is not the case for Windows. We have begun exploring this approach but need time to develop it. A less ambitious approach could be to modify individual software applications to provide individual deceptions. This would allow us to more easily explore a diversity of software functions and more easily test the human-factors issues.

We explored this idea in experiments with a prototype software module providing simple deceptive responses to protect a World Wide Web site [3]. We examined three methods of responding to a malicious attempt to overflow the input buffer. All were done by modifying an image-browser Web portal (interface program) we wrote that was implemented with the Java "servlet" package. The original portal indexes most of the images on U.S. military ("*.mil") sites; it uses the results of a page crawler that employs heuristics to rank likelihood of text being a caption of an image. A user enters a set of keywords describing images that he or she is looking for, and the portal retrieves the images from the Web that it thinks best match the keywords. Changes were made to the code to permit deceptive behavior once preconditions of suspiciousness are met, while treating normal users as before. In deceptive mode the operating system is much less likely to be attacked because it is being simulated in a safe environment (a "sandbox").

One deceptive tactic involved applying a random delay to responses of the portal. This makes it appear that the input of the malicious user is slowing down the computer system, a desired effect of a denial-of-service attack. We implement this by delaying normal responses enough to make them approximate a fixed multiple of the average response time during normal conditions. The multiplier was estimated from the number of current request transactions and the work required to process each keyword of the request. Rules triggered delaying whenever: (1) keywords began with "file//", suggesting an attempt to access arbitrary files on the server; (2) keywords resembled instructions in the language C in the use of "=" and "+", suggesting an attempt to insert code; (3) keywords began with "//", suggesting escape-character sequences for sending commands directly to the operating system; (4) there was just one long keyword, suggesting code insertion; or (5) there were more than 10 keywords, suggesting denial-of-service attacks. Delays were accomplished with the Java process-suspension mechanism, and their time included a random factor to avoid being too predictable.

A second tactic that we explored simulated a login screen in response to suspicious behavior. There were two variations on this deception, a login-window simulation and a "root shell" simulation. The first was a popup window that prompted the user for their name and password, offering "OK" and "Cancel" buttons, but not actually doing anything further. The root-shell approach simulated a root (system-administrator) shell like that of "Command Prompt" in Microsoft Windows. The idea of both was to delay the user in a more interactive way than just using process suspension, keeping them occupied for a while at useless tasks. Both used the same

triggers described above.

We conducted experiments with eight test subjects, graduate students at our school not familiar with our research. The subjects were individually placed in front of a computer displaying the portal Web page as modified with deceptive code. They were told the keyword box would accept their input, and they were asked to type up to five words for the program to search for. After they completed two searches, they were told that the mode they were operating in was considered ?normal.? They were then asked to enter strings that the deception programs considered suspicious. Subjects were then asked to provide an overall rating of the success of the deception, on a scale of 1 to 5, as well as whether they themselves were fooled as to whether the system was acting normall;y.

The subjects had a wide range of computer-related experience, but all reported being fooled by the deception, especially the delaying tactic. While most subjects blindly estimated the processing time before the execution of the first search, the program successfully accounted for the processing time by proving valid our hypothesis that time sequences are perceived well by users but not durations of events in a sequence.

The simulated login screen and the fake root shell generated better-than-expected reactions. Six of eight subjects felt the appearance of the screens was surprising and believable. The root-shell simulation was not expected by any subjects, and successfully surprised them. The reactions among the computer-science students were especially noteworthy because their subsequent interaction with the shell seemed to match their expectations for normal response of the system to their requests. So for their commands, the ?*Command completed successfully*? message provided sufficient confirmation to be believable.

The overall believability of the responses was high, averaging 4.6 out of a possible 5.0 where 4 was "believable" and 5 was "very believable". One subject stated he was ?sort of fooled? by the tactics used, but all other subjects stated they were for the most part fooled. The only skepticism was with the more experienced subjects who thought the simulated screens were too easy to obtain. However, these subjects expressed favorable remarks at the realism of the display, especially when integrated with the delaying method. Responses to delaying tactics also confirmed the hypothesis that subjects were not concerned with the time it took to process their malicious request, only that it took longer than normal. A second goal of the experiment was to determine if the subjects could tell they were being deceived, and all subjects did believe the computer was processing their malicious request normally.

Future work will include more complex levels of responses, methods of isolation of the deceived attacker from the system under attack, and methods for integrating deception into software. Other work of ours has already produced a standalone deceptive file-transfer utility that simulates some published vulnerabilities [4]. Our research into software-based deception fits between the capabilities of current intrusion-detection techniques and potential counteractive techniques, to better equip computer systems for the next generation of cyber-warfare by providing new kinds of responses to intrusions.

REFERENCES

[1] Rowe, N. C., Michael, J. B., Auguston, M., and Riehle, R., Software decoys for software counterintelligence. *IAnewsletter,* Vol. 5, No. 1 (Spring 2002), pp. 10-12.

[2] Michael, J.B., Auguston, M., Rowe, N., and Riehle, R.D., Software decoys: intrusion detection and countermeasures. Proc. Third Annual Workshop on Information Assurance, IEEE, West Point, New York, June 2002, pp. 130-138.

[3] Julian, D. P., Delaying-type responses for use by software decoys. M.S. thesis, Dept. of Computer Science, U.S. Naval Postgraduate School, September 2002, http://www.cs.nps.navy.mil/people/faculty/rowe/oldstudents/Julian_Thesis_Final.htm.

[4] Michael, J. B., Fragkos, G., and Auguston, M., An experiment in software decoy design: intrusion detection and countermeasures. Proc. 18th IFIP International Information Security Conference, Athens, Greece, May 2003.

Acknowledgement: This work is part of the Homeland Security Leadership Development Program supported by the U.S. Department of Justice Office of Justice Programs and Office for Domestic Preparedness. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government. *This paper appeared in the Proceedings of the 2003 IEEE Workshop on Information Assurance, West Point, NY, June 2003.*